第5章 chapter 5

数　组

数组是数据的集合，是将若干具有相同数据类型的变量按有序形式组织起来，在内存中连续存放，便于数据的存取。

当有大量数据需要批量存储及处理时，数组便是很好的选择。本章主要介绍一维数组、二维数组的定义、初始化、元素引用等基本概念，以及数组的遍历、查找、排序等编程方法。

内容导读：

- 掌握一维数组、二维数组的定义、初始化、元素引用等基本概念。
- 理解数组名的特殊含义。
- 掌握对数组元素求最大值、最小值、查找、排序等常用算法。

5.1　为何要使用数组

假设有10名学生，每名学生有一门课成绩需要存储，根据已学知识，需定义10个变量进行存放，例如，“int score1，score2，…，score10；”，但如果有100、1000门课成绩需要存储，难道要依次定义100个，甚至1000个变量吗？显然，程序设计不应该如此烦琐。为便于对类似这样的批量数据进行处理，C语言提供了**数组(array)**的概念。

对于10名学生，每名学生一门课成绩，共有10个数据，可定义一个一维数组对其进行存储，如“int score[10]；”。而如果每名学生有5门课成绩，共有50个数据，则可以定义一个二维数组对其进行存储，如“int score[10][5]；”。

以上是数组的应用场景，定义数组可一次性批量定义若干相同类型的变量，这些变量称为数组元素，编译系统为这些变量批量分配内存单元，它们在内存中占用一段连续的存储空间。地址的连续性使得程序对批量数据的访问及处理变得较为容易。本章将重点介绍一维数组和二维数组的定义及使用方法。

5.2　一维数组

5.2.1　一维数组定义

一维数组是按序排列的同类数据元素的集合。如前所述，一维数组定义即表示批量

定义若干类型相同的变量，以下为其定义格式：

数据类型　数组名[整型常量表达式];

例如，“int score[10];”表示定义了一个大小为10的整型数组，数组名为score，该数组中包含10个整型变量。

敲重点：

(1) 一维数组定义即表示批量定义了若干同类型的变量，将这些变量称为数组元素(element)。

(2) 数据类型指定了所有数组元素共同的类型。可以是基本类型，如int、char、float、double等，也可以是第7、9章中介绍的指针类型、结构体类型等。

(3) 数组名是所有数组元素共同的标识，属于用户标识符，应遵循标识符的命名规则。特别地，所有数组元素在内存中占用一段连续的内存空间，数组名代表了这段内存空间的起始地址，因此数组名是地址常量。

(4) 数组大小由整型常量表达式指定，即数组元素的个数。数组一旦定义，其大小将不能改变。

5.2.2 一维数组元素引用

一维数组定义后，其中的数组元素是变量，可存放数据，对数组元素的引用即是对数据的访问，以下为数组元素的引用格式：

数组名[下标]

假设有定义“int score[10];”，数组中包含10个int型变量，依次为score[0]、score[1]、……、score[9]，系统为该数组分配40B的连续存储空间，每个数组元素占4B，其存储结构如图5-1所示。

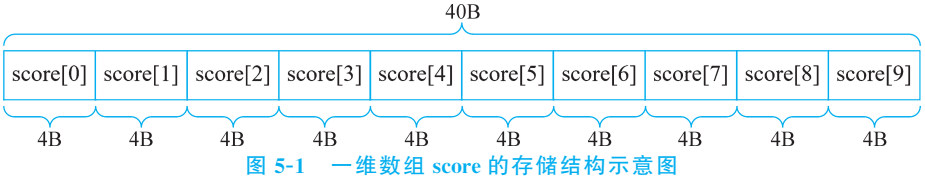

图5-1　一维数组score的存储结构示意图

敲重点：

(1) 数组元素是变量，用于存放数值。

(2) 数组元素的下标(subscript)从0开始，最大下标为“元素个数－1”。下标确定了该数组元素在数组中的序号，可以是整型常量、变量、表达式。

【例5.1】 定义一个大小为5的整型数组，输入5个分数存入数组中并求总分。

思路：假设数组定义为“int a[5];”，输入的5个分数将依次存放到数组元素a[0]～a[4]中，利用数组元素下标的连续性，可使用循环结构对数组元素依次进行访问。

一维数组定义及元素引用

```
1.  #include <stdio.h>
2.  int main()
3.  {
```

```
4.      int a[5], i, sum = 0;                  //注意:求和变量 sum 的初值应为 0
5.      printf("请输入 5 个分数:");
6.      for(i = 0; i < 5; i++)                 //循环 5 次,遍历 5 个数组元素
7.      {
8.          scanf("%d", &a[i]);                //将每次输入的整数放到数组元素 a[i]中
9.          sum += a[i];                       //求总分
10.     }
11.     printf("总分: %d\n", sum);
12.     return 0;
13. }
```

程序运行结果：

```
请输入5个分数：78 90 82 95 84
总分: 429
```

分析：for 循环变量 i 的取值范围是[0，4]，变量 i 有两个作用，一是控制循环次数，二是作为数组元素的下标(符合下标从 0 开始的语法规定)。

5.2.3 一维数组初始化

一维数组初始化是指**在定义数组时为数组元素赋初值**。以下为一维数组的初始化格式：

```
数据类型  数组名[整型常量表达式] = {初值表};
```

一维数组的初始化主要有以下四种形式。

(1) 全部元素初始化。

例如：

```
int a[5] = {10, 20, 30, 40, 50};
```

注意：初值的个数不能超过数组大小。

(2) 全部元素初始化，可缺省数组大小。

例如：

```
int a[ ] = {10, 20, 30, 40, 50};
```

编译系统根据花括号内的初值个数可确定数组大小为 5。

(3) 部分元素初始化。

例如：

```
int a[5] = {10, 20, 30};
```

花括号内仅给定 3 个数值，而数组大小为 5，于是编译系统自动将 a[3]、a[4]赋初值为 0。

(4) 所有数组元素赋初值为 0。

例如：

```
int a[5] = {0};
```

以上是数组元素a[0]～a[4]均赋初值为0的一种方法。需要说明的是，如果数组定义后未赋初值，则所有元素的初值是随机数。

【例5.2】 定义大小为10的整型数组，初始化所有数组元素，查找数组中的最大值。

思路：查找最大值的方法是，首先默认第一个数最大，将其放入max变量中，然后用max的值与后续元素依次比较，每次比较总是将较大值放入max中，如此当所有元素比较完毕后，max中即存放数组中的最大值。


顺序法查找最大值

```
1.  #include <stdio.h>
2.  int main()
3.  {   //初始化数组元素
4.      int a[10] = {85, 92, 78, 66, 95, 70, 80, 82, 56, 88}, i, max;
5.      printf("数组初值:");
6.      for(i = 0; i < 10; i++)
7.      {
8.          printf("%-6d", a[i]);
9.      }
10.     max = a[0];                     //默认第一个数 a[0]最大,暂存于变量 max 中
11.     for(i = 1; i < 10; i++)         //依次比较后续元素
12.     {
13.         if(a[i] > max)              //判断 a[i]是否大于 max 中的值
14.         {
15.             max = a[i];             //将找到的更大的值放入 max 中
16.         }
17.     }
18.     printf("\n最大值:%d\n", max);
19.     return 0;
20. }
```

程序运行结果：

```
数组初值：85    92    78    66    95    70    80    82    56    88
最大值：95
```

分析：关于数组求最大值的问题，初学者容易将程序第13行的if判断条件误写成if(a[i] > a[i+1])。注意，求最大值不是对相邻数组元素进行比较，而是预设一个保存最大值的变量max，用max依次与各数组元素进行比较，每次比较结束后，总是将较大值存入max中。

5.2.4 一维数组常见错误

1. 数组定义的常见错误

(1) 数组名是地址常量，不是变量。

例如：

```
int score[10];
score = 80;                         //错误
```

错误原因：数组名score是地址常量，不能被赋值，只有变量才能被赋值。

(2) 数组定义时，必须用常量值标识数组的大小。

例如：

```
int m = 10;
int a[m];                                    //错误
```

错误原因：数组定义时，方括号内必须为整型常量表达式，用于标识数组的大小。而定义语句“int a[m];”中的 m 是变量，不符合语法规定。

2. 数组元素引用的常见错误

数组元素的下标范围是 0～数组大小－1，如果下标超过此范围，称为下标越界。编译系统通常不会对下标越界报错，但是下标越界会导致访问无效的数组元素，从而引起程序运行错误。

【例 5.3】 数组元素下标越界导致程序运行出错。


数组元素下标越界的问题

```
1.  #include <stdio.h>
2.  int main()
3.  {
4.      int a[5], i;
5.      a[0] = 10;     a[1] = 20;     a[2] = 30;     a[3] = 40;     a[4] = 50;
6.      printf("数组元素: \n");
7.      for(i = 0; i <= 5; i++)
8.      {
9.          printf("%d\t", a[i]);
10.     }
11.     return 0;
12. }
```

程序运行结果：

```
数组元素:
10      20      30      40      50      0
```

分析：

(1) 程序第 5 行是对数组元素分别赋值，数组元素 a[0]～a[4] 是 5 个独立的变量，不能整体赋值，需分别赋值。

(2) 程序第 7 行的 for 语句循环了 6 次，错误引用了 a[5]，而 a[5]是无效元素，但是编译器并不会对下标越界给出错误提示，程序设计者需自行检查数组元素下标的合法性。

5.2.5 一维数组应用举例

【例 5.4】 使用一维数组编程，求斐波那契(Fibonacci)数列的前 20 项。斐波那契数列定义：第一个数是 1，第二个数是 1，从第三个数开始，每个数等于前两个数之和。可以用数学上的递推公式来表示：

$$f(n)=\begin{cases}1 & (\text{当 } n=1 \text{ 或 } n=2 \text{ 时})\\ f(n-1)+f(n-2) & (\text{当 } n>2 \text{ 时})\end{cases}$$

思路：斐波那契数列的前几项为 1、1、2、3、5、8、13、21、34……根据题意，本题可定义一个大小为 20 的整型数组 a，为数组元素 a[0]、a[1]赋初值 1、1，再循环 18 次，依次求出

a[2]～a[19]的值。


求斐波那契数列

```
#include <stdio.h>
#define  N  20
int main()
{
    int a[N] = {1, 1}, i;                    //仅初始化数组元素 a[0]、a[1]
    for(i = 2; i < N; i++)
    {
        a[i] = a[i-1] + a[i-2];
    }
    printf("斐波那契数列的前%d项: \n", N);
    for(i=0; i<N; i++)
    {
        printf("%d\t", a[i]);
    }
    return 0;
}
```

程序运行结果：

```
斐波那契数列的前20项:
1       1       2       3       5       8       13      21      34      55
89      144     233     377     610     987     1597    2584    4181    6765
```

【例 5.5】 用冒泡排序法将 N 个整数按照由小到大的顺序(升序)进行排序。

冒泡排序法思想(以升序排序为例)：对待排序的数组，依次两两比较相邻元素，如果前面的数大于后面的数，则交换两数，不断重复以上过程，直至排序结束。

假设有 5 个数 3，5，4，2，1，使用冒泡法对它们进行升序排序，排序过程如表 5-1 所示。表中将每次被比较的相邻元素加框显示。

表 5-1 冒泡法对 5 个数排序的过程

排序的轮数	本轮等待排序的数	每轮排序中的比较过程				本轮排序的结果
		第 1 次	第 2 次	第 3 次	第 4 次	
第 1 轮	3 5 4 2 1	[3 5] 4 2 1	3 [4 5] 2 1	3 4 [2 5] 1	3 4 2 [1 5]	5 被放到队列第五的位置
第 2 轮	3 4 2 1	[3 4] 2 1 5	3 [2 4] 1 5	3 2 [1 4] 5		4 被放到队列第四的位置
第 3 轮	3 2 1	[2 3] 1 4 5	2 [1 3] 4 5			3 被放到队列第三的位置
第 4 轮	2 1	[1 2] 3 4 5				2 被放到队列第二的位置

由表 5-1 可知，共进行了 4 轮排序，每轮排序又进行若干次比较。可使用双层循环嵌套实现，外循环控制排序的轮数，内循环控制每轮排序中比较的次数。假设外循环变量是 i，内循环变量是 j，则 i、j 的取值范围见表 5-2 的分析。

表 5-2 冒泡排序过程中内、外循环变量取值范围的分析

排序的轮数	外循环变量 i 的取值	每轮排序中依次比较相邻两数				内循环次数/次	内循环变量 j 的取值
		第 1 次比较	第 2 次比较	第 3 次比较	第 4 次比较		
第 1 轮	i = 0	a[0]和 a[1]	a[1]和 a[2]	a[2]和 a[3]	a[3]和 a[4]	4	j：0～3
第 2 轮	i = 1	a[0]和 a[1]	a[1]和 a[2]	a[2]和 a[3]		3	j：0～2

续表

排序的轮数	外循环变量 i 的取值	每轮排序中依次比较相邻两数				内循环次数/次	内循环变量 j 的取值
		第 1 次比较	第 2 次比较	第 3 次比较	第 4 次比较		
第 3 轮	i = 2	a[0]和 a[1]	a[1]和 a[2]			2	j：0～1
第 4 轮	i = 3	a[0]和 a[1]				1	j：0

由表 5-2 可知，外循环 4 次，外循环变量 i 的取值范围是 0～3；内循环次数逐渐递减，可理解为内循环次数随着外循环变量 i 的增大而减少，于是内循环变量 j 的取值范围是 0～3－i。

冒泡排序法程序如下：


冒泡排序

```
#include <stdio.h>
int main()
{
    int a[5] = {3, 5, 4, 2, 1}, i, j, t;
    printf("(1)排序之前的数: ");
    for(i = 0; i < 5; i++)
    {
        printf("%-4d", a[i]);
    }
    for(i = 0; i < 4; i++)          //外循环控制排序的轮数
    {
        for(j = 0; j < 4 - i; j++)  //内循环控制每轮排序比较的次数
        {
            if(a[j] > a[j+1])       //a[j]和 a[j+1]是被比较的相邻两数
            {
                t = a[j];           //用三条赋值语句实现 a[j]和 a[j+1]的交换
                a[j] = a[j+1];
                a[j+1] = t;
            }
        }
    }
    printf("\n(2)升序排序后的数: \n");
    for(i = 0; i < 5; i++)
    {
        printf("%-4d", a[i]);
    }
    return 0;
}
```

程序运行结果：

```
(1)排序之前的数:  3   5   4   2   1
(2)升序排序后的数: 1   2   3   4   5
```

5.3 二维数组

5.3.1 二维数组定义

假设有 3 名学生，每名学生有 4 门课成绩，共 12 个分数，如何存储更为合理？联想用

Excel表存储学生成绩的格式，通常为每行对应一名学生，每列对应一门课程。类似于生活中这种使用矩阵结构存储数据的形式，在C语言中就是二维数组。

以下为二维数组定义的格式：

数据类型　数组名[整型常量表达式1][整型常量表达式2];

例如，“int a[3][4];”表示定义了一个3行4列共12个元素的二维数组，数组名为a。

敲重点：

（1）二维数组定义时，数组名后面有两对方括号，方括号内都是整型常量表达式，分别指定二维数组的行长度和列长度。

（2）二维数组定义与一维数组定义相同，都是批量定义若干同类型的变量，将这些变量称为二维数组元素，它们在内存中按行存放。

5.3.2　二维数组元素引用

以下为二维数组元素引用的格式：

数组名[行下标][列下标]

假设有定义“int a[3][4];”，则二维数组a中包含12个元素，其存储结构如图5-2所示。

	第1列元素	第2列元素	第3列元素	第4列元素
第1行元素	a[0][0]	a[0][1]	a[0][2]	a[0][3]
第2行元素	a[1][0]	a[1][1]	a[1][2]	a[1][3]
第3行元素	a[2][0]	a[2][1]	a[2][2]	a[2][3]

图5-2　二维数组a的存储结构示意图

敲重点：

（1）二维数组元素的行、列下标取值都是从0开始。

（2）观察图5-2可知，对于二维数组的同一行元素，其行下标相同，列下标不同；而对于同一列元素，其列下标相同，行下标不同。

（3）二维数组元素由行和列组成，也可以将二维数组看作一个特殊的一维数组，其元素的排列顺序如图5-3所示。

a[0][0]	a[0][1]	a[0][2]	a[0][3]	a[1][0]	a[1][1]	a[1][2]	a[1][3]	a[2][0]	a[2][1]	a[2][2]	a[2][3]

图5-3　二维数组元素在内存中按行存放的示意图

5.3.3　二维数组初始化

二维数组的初始化与一维数组类似，用花括号将初值括起来。由于二维数组既可以理解为行列矩阵的形式，其各行元素按行存放；又可以理解为是一个特殊的一维数组，其各行元素顺序存放，于是二维数组的初始化有分行初始化和不分行初始化两种形式。

(1) 分行初始化——对全部元素赋初值。

例如：

```
int a[3][4] = {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}};
```

以上写法用两层花括号直观地表示了所有数值与各行元素的对应关系，其中内层有3对花括号，对二维数组3行元素的初值进行了界定。

(2) 分行初始化——对部分元素赋初值。

例如：

```
int b[3][4] = {{1, 2}, {3, 4, 5}, {6}};
```

以上写法仅给定了部分元素的初值，而对于未赋值的数组元素，系统自动赋初值为0。这些默认赋初值为0的数组元素分别是b[0][2]、b[0][3]、b[1][3]、b[2][1]、b[2][2]、b[2][3]。

(3) 不分行初始化——对全部元素赋初值。

例如：

```
int c[3][4] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};
```

以上写法将12个初值根据按行存放的原则，依次对应赋初值给12个数组元素。

(4) 不分行初始化——对部分元素赋初值。

例如：

```
int d[3][4] = {1, 2, 3, 4, 5, 6};
```

以上写法仅对数组前面的6个元素给定初值，后面的8个元素系统自动赋初值为0。

(5) 二维数组定义时，可以缺省数组的行长度表达式，编译系统将根据初始化的数值个数自动确定二维数组的行长度。

例如：

```
int e[ ][3] = {{1, 2, 3}, {5, 6}};        //此时系统自动确定二维数组为2行3列
int f[ ][4] = {1, 2, 3, 4, 5, 6, 7, 8};   //此时系统自动确定二维数组为2行4列
```

注意：语法规定只能缺省二维数组的行长度，不能缺省列长度。

5.3.4 二维数组应用举例

由图5-2可知，二维数组元素可视为由行和列组成，同一行元素的行标相同，同一列元素的列标相同，于是**对二维数组元素的访问有两种方法：按行遍历和按列遍历。**

使用双层循环嵌套可实现对二维数组的遍历，如表5-3所示。

表5-3 二维数组的遍历

遍历方法	特　　点	外循环	内循环
按行遍历	以行为单位对二维数组元素进行访问	控制行	控制列
按列遍历	以列为单位对二维数组元素进行访问	控制列	控制行

【例5.6】 有3名学生，每名学生有4门课(高数、C语言、政治、英语)的成绩，定义一

个二维数组存放学生成绩，编程计算每名学生的平均分，以及每门课程的平均分。

思路：计算每名学生的平均分，需对每名学生的4门课成绩求总分，可使用按行遍历的方法访问二维数组。计算每门课程的平均分，需对每门课程的3名学生成绩求总分，可使用按列遍历的方法访问二维数组。


二维数组处理成绩

```
#include <stdio.h>
int main()
{
    int s[3][4] = {{64, 72, 78, 66}, {88, 95, 94, 85}, {78, 82, 87, 80}};
    int i, j, sum;
    printf("----------3名学生4门课的成绩----------\n");
    printf("\t高数\tC语言\t政治\t英语\n");
    for(i = 0; i < 3; i++)              //按行访问二维数组
    {
        printf("学生%d\t", i+1);
        for(j = 0; j < 4; j++)
        {
            printf("%d\t", s[i][j]);
        }
        printf("\n");
    }

    printf("\n3名学生的平均分:");
    for(i = 0; i < 3; i++)              //按行访问二维数组
    {
        sum = 0;
        for(j = 0; j < 4; j++)
        {
            sum += s[i][j];
        }
        printf("%.2f\t", sum / 4.0); //每名学生的总分除以课程门数得学生平均分
    }

    printf("\n4门课程的平均分:");
    for(i = 0; i < 4; i++)              //按列访问二维数组
    {
        sum = 0;
        for(j = 0; j < 3; j++)
        {
            sum += s[j][i];
        }
        printf("%.2f\t", sum / 3.0); //每门课程的总分除以学生人数得课程平均分
    }
    return 0;
}
```

程序运行结果：

```
----------3名学生4门课的成绩----------
        高数    C语言   政治    英语
学生1   64      72      78      66
学生2   88      95      94      85
学生3   78      82      87      80

3名学生的平均分：70.00  90.50   81.75
4门课程的平均分：76.67  83.00   86.33   77.00
```